Ansteuerung einer Parallelport-Webcam mit CPiA Videoprozessor

Studienarbeit

Thorsten Geier

Betreuer:
Dr. Merten Joost

01 April 2005
Sommersemester 2005

Institut für integrierte Naturwissenschaften, Abteilung Physik
Universität Koblenz-Landau
Universitätsstraße 1, 56070 Koblenz

Inhaltsverzeichnis

1 Einleitung 4
1.1 Motivation 4
1.2 Aufgaben und Ziele 5
1.3 Entwicklungsumgebung 5
2 Technische Grundlagen 6
2.1 Webcam Beschreibung 6
2.2 Inbetriebnahme der Kamera 8
3 Überblick über die parallele Schnittstelle 9
3.1 IEE1284 Parallelport Modi 10
3.1.1 Compatible Mode 10
3.1.2 Nibble Mode 11
3.1.3 Byte Mode 11
3.1.4 Extended Parallel Port 11
3.1.5 Enhanced Capability Mode 12
3.2 Auswahl des Übertragungsmodus 13
3.3 Zugriff auf den Parallelport unter Windows 14
3.4 Portzugriffe mit Java 15
3.5 Portzugriffe mit C/C++ 16
3.6 Installation und Benutzung von Inpout32.dll 16
3.7 Kontrollwerkzeug für Parallelportzugriffe 18
4 Der CPiA Prozessor 18
4.1 Videoprozessor 19
4.2 Videokompressionsmodul 20
4.3 Kontrollprozessor 21
5 Ansteuerung der Kamera 22
5.1 Vorhandener Quellcode 22
5.2 PP_test 22
5.2.1 pport.c, comm.c und cpialink.c 22
5.2.2 cmd.h und cpia_cmd.c 23
5.2.3 cconvert.c, decode.c und cpia_vid.h 23
5.2.4 main_menu.c 28
5.2.5 pp_test.c 32
5.3 SmallVersionECP 33
5.4 CamMonitor 33
5.4.1 imageviewer.cpp 34
5.4.2 thread.cpp 35
5.5 XsmallVersion 36
5.5.1 SendPacket( ) 37
5.5.2 GetPacket( ) 38
5.5.3 UploadStreamData( ) 39
6 Fazit und Ausblick 40

Abbildungsverzeichnis

Abbildung 1: Technische Daten *7*
Abbildung 2: Creative Video Blaster WebCam II *7*
Abbildung 3: Geräte-Manager *8*
Abbildung 4: Kamerasoftware von Ceative *9*
Abbildung 5: Nibble Modus *10*
Abbildung 6: Byte Modus *11*
Abbildung 7: EPP Modus *12*
Abbildung 8: ECP Modus *12*
Abbildung 9: IEEE1284 Modi *13*
Abbildung 10: Fehlermeldung unter Windows XP *14*
Abbildung 11: Initialisierung von inpout32.dll *16*
Abbildung 12: Ablaufdiagramm von inpout32.dll *17*
Abbildung 13: Deinitialisierung der inpout32.dll *17*
Abbildung 14: Parallel port monitor *18*
Abbildung 15: CPiA System Block Diagramm *19*
Abbildung 16: Die CPiA Bilddatenverarbeitung *19*
Abbildung 17: Das CPiA Videokompressionsmodul *20*
Abbildung 18: Benutzte Parallelport-Pins *21*
Abbildung 19: YUVPixeltoRGB24( ) *24*
Abbildung 20: Format ToSupplyFourCC( ) *26*
Abbildung 21: Bildgrößen ohne Kompression *26*
Abbildung 22: Frameheader *27*
Abbildung 23: SingleGrab( ) *29*
Abbildung 24: Erstes Bild ohne geänderter Pixelfolge und vertauschten RB-Werten *29*
Abbildung 25: SaveBufferToFile( ) *30*
Abbildung 26: ContinousGrab( ) *31*
Abbildung 27: Aufnahmezeiten für ContinuousGrab( ) *31*
Abbildung 28: StreamGrab( ) *32*
Abbildung 29: SmallVersionECP Benutzermenü *33*
Abbildung 30: CamMonitor mit grafischer Benutzeroberfläche *34*
Abbildung 31: Thread zur Bildaufnahme *36*
Abbildung 32: Der Befehl GetCPiAVersion *36*
Abbildung 33: SendPacket( ) *37*
Abbildung 34: WriteBuffer( ) *38*
Abbildung 35: ECPReadBuffer( ) *38*
Abbildung 36: UploadStreamData( ) *39*

1 Einleitung

1.1 Motivation

Webcams sind mittlerweile sehr weit verbreitet und werden z.B. für Videochat- und Videotelefonie benutzt. Durch ihre einfache Inbetriebnahme und Handhabung sind sie unter Computernutzern sehr beliebt. Größter Nachteil der Webcams ist, dass man sie nur dann zur Bild- oder Videoaufnahme benutzen kann, wenn sie an einen Computer angeschlossen sind und dieser auch eingeschaltet ist. Zur Raumüberwachung für eine Alarmanlage bieten sich einfache Webcams daher nicht an. Für eine Webcam die kontinuierlich Bilder aufnimmt und an einen Webserver schickt, braucht man immer einen Computer der die Kamera ansteuert, die Bilder entgegen nimmt und diese dann an den Webserver weiterleitet. Für diesen Einsatzzweck gibt es Webcams mit Netzwerkanschluss, sog. Network Cameras oder Netcams. Diese Kameras lassen sich direkt an ein Netzwerk anschließen und über eine eigene IP-Adresse ansteuern. Teilweise haben die Netzwerkkameras eine RISC-CPU mit Speicher und einem Mini-Linux als Betriebssystem „an Bord“. Diese Kameras können dann direkt als Web-Server fungieren. Neben dem Anschluss über 10BaseT / 100BaseTX Ethernet über eine RJ-45 Buchse, gibt es auch NetCams die über WLan kommunizieren. Dies ermöglicht den mobilen Einsatz der NetCams, u.a. auch in Modellbauflugzeugen oder Autos. Angedacht ist der Bau eines (autonomen) Fahrzeugs, das mit einer Netcam ausgestattet ist und somit Bilder von seiner Umgebung übertragen kann.

Im Vergleich zu einfachen Webcams sind Netcams wesentlich teurer in der Anschaffung. Je nach Funktionsumfang ergibt sich ein Aufpreis von ca. 100 bis 200 Euro. Die hohen Anschaffungskosten können als ein Grund angesehen werden, eine Netcam selbst zu bauen. Basis zur Realisierung einer selbstgebauten Netcam sind eine Webcam und ein günstiger programmierbarer Controller mit einer Netzwerk- und einer weiteren Schnittstelle zum Anschluss der Webcam. Alternativ zu heutigen Webcams mit USB-Anschluss kommen für dieses Projekt auch ältere Webcams zum Anschluss an den Parallelport in Betracht, da sich die Ansteuerung des Parallelports im Controller leichter realisieren lässt.

Getestet wird die Realisierung mit einer Video Blaster WebCam II von Creative Labs. Damit die Kamera am Parallelport des Controllers steuerbar ist, muss aber zunächst ermittelt werden, wie die Kamera überhaupt angesprochen wird. Mit diesem Teilproblem beschäftigt sich diese Studienarbeit im Folgenden.

1.2 Aufgaben und Ziele

Die generelle Zielsetzung der Studienarbeit ist es, Wissen über die Ansteuerung der verwendeten Creative VideoBlaster II Kamera und deren CPiA-Controller zu sammeln. Neben der Inbetriebnahme der Kamera und dem Ansprechen des Parallelports liegt der Schwerpunkt der Studienarbeit in der Programmierung einer Software zum Steuern der Kamera. Die Software sollte die Möglichkeit bieten die grundlegenden Fähigkeiten der Kamera zu nutzen. Dazu gehört die Aufnahme von Bildern und die Übertragung der Bilder an den Computer. Alle dafür notwendigen Aufnahmeparameter sollten ebenfalls von der Software mit CPiA-Befehlen gesetzt werden können. Zusätzlich sollte die erstellte Software in der Lage sein, die von der Kamera übermittelten Bildinformationen zu verarbeiten. Die übertragenen Bilder sollten dargestellt oder in einem lesbaren Bildformat gespeichert werden können.
Im Hinblick auf die Tatsache, dass die Kamera später direkt an einen Controller angeschlossen werden soll, sollte die Ansteuerung der Kamera so beschaffen sein, dass sie sich später mit wenig Aufwand auf den Controller übertragen lässt.

1.3 Entwicklungsumgebung

Zum Einsatz kommt ein Computer mit Pentium 4 Prozessor und 2 GHz Prozessortakt mit WindowsXP als Betriebssystem. Wobei die Prozessorgeschwindigkeit für die Studienarbeit vernachlässigbar ist. Wichtiger sind das Vorhandensein von PS2-Tastaturanschluß und der Druckerport zum Anschluss der Kamera. Als Programmierumgebungen kommen für Java Eclipse und für C/C++ VisualStudio und DevC++ zum Einsatz. Für die Entwicklung einer grafischen Benutzeroberfläche zusätzlich noch eine Testversion von Qt 4.0 des Herstellers Trolltech.

2 Technische Grundlagen

2.1 Webcam Beschreibung

Die verwendete Video Blaster WebCam II von Creative Labs stammt aus dem Jahr 1998. Da mittlerweile zahlreiche Nachfolgemodelle herausgekommen sind, ist diese Kamera fast nur noch gebraucht zu bekommen. Als Bezugsquelle dient das Online-Auktionshaus Ebay. Die Preise liegen bei etwa einem bis fünf Euro pro Kamera.

Zu den technischen Daten:

Anschluss:	25 pol. Sub-D zum Anschluss an Parallel-Port-Schnittstelle
Stromversorgung:	Über DIN oder PS2 Tastatur Adapter
Sensor:	Vision VM6426 ColorMOS Bildsensor
Auflösung:	Videomodus: 160 x 120 176 x 144 320 x 240 352 x 288 Standbildmodus: 160 x 120 176 x 144 320 x 240 352 x 288 640 x 480 (interpoliert) 704 x 576 (interpoliert)
Empfindlichkeit:	15 lux bei 7,5 fps
Geschwindigkeit:	max. 20 fps im 352 x 288 Video Mode max. 30 fps im 176 x 144 Video Mode
Video Formate:	24- und 16-Bit RGB 8-Bit RGB (mit Farbpalette) 4:2:0 YUV planar YUV2 UYVY YUV9
Belichtung:	automatisch (oder manuell per Software)

Farbbalance:	automatisch (oder manuell per Software)
Blickwinkel:	52° (horizontal)
Tiefenschärfe:	50 Millimeter bis unendlich
Port-Kompatibilität:	Standard- (unidirektionaler) Drucker-Port, 8-Bit bidirektional, ECP-Enhanced-Modus

Abbildung 1: Technische Daten

Das Gehäuse der Kamera ist recht kompakt. Die Kamera besitzt keinen Autofokus. Zur Einstellung der Fokussierung muss die Objektivlinse per Hand gedreht werden. Für den Anschluss an einen aktuellen PC muss dieser über einen Druckerport und eine PS2-Tastatur-Buchse verfügen. Benötigt wird dafür noch ein extra Adapter von DIN auf PS2. Ist keine PS2- Tastatur-Buchse verfügbar, lässt sich die Kamera auch mit einer externen Spannungsversorgung betreiben. Die Gleichspannung sollte 5V betragen und kann über Pin X und Pin Y des am Kabel befestigten PS2 Steckers oder über Pin X und Pin Y des DIN Steckers eingespeist werden.

Ein Hinweis für die externe Stromversorgung:
Die Kameraelektronik reagiert recht empfindlich auf „unsaubere“ Gleichspannung, wie sie z.B. unstabilisierte Steckernetzteile liefern, dadurch kann es zu unerwünschten Bildstörungen kommen.

Abbildung 2: Creative Video Blaster WebCam II

2.2 Inbetriebnahme der Kamera

Zum Anschluss der Kamera kann man sich an die Anweisungen im mitgelieferten Handbuch halten. Auf der Treiber-CD von Creative sind Treiber für Windows95, Windows98 und WindowNT enthalten. Damit die Kamera von WindowsXP erkannt wird, installiert man am besten den Treiber für WindowsNT. Bei erfolgreicher Installation erscheint die Kamera im Gerätemanager von Windows als VLSI Vision Ltd PPC2 Camera.

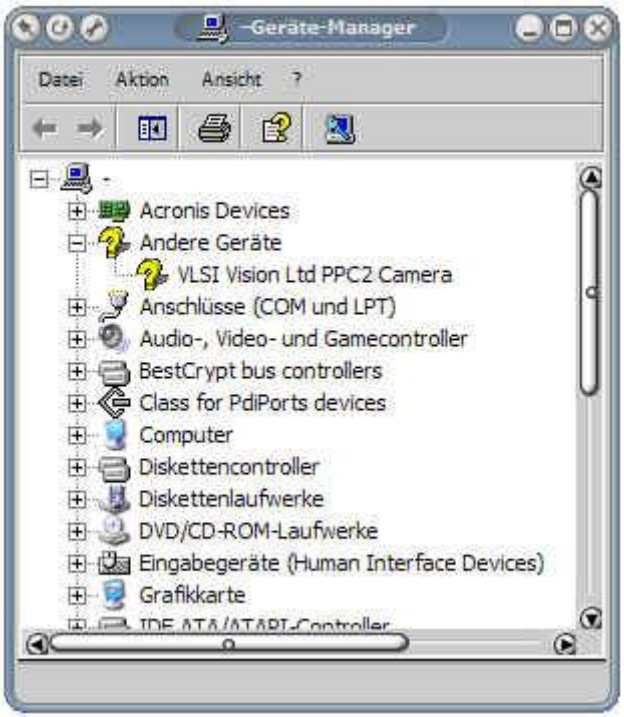

Abbildung 3: Geräte-Manager

Hilfreich zur Funktionsüberprüfung der Kamera ist eine Installation der ebenfalls auf der CD enthaltenen Kamerasoftware. Sollte die Kamera nach einer experimentellen, bzw. nicht ganz korrekten Ansteuerung nicht mehr reagieren, kann der Start des Programms „WebCam Monitor“ die Kamera ggf. reaktivieren. Weiterhin lassen sich die Kamerabilder und die Anzeigegeschwindigkeit gut mit dem eigenen Programm zur Kameraansteuerung vergleichen.

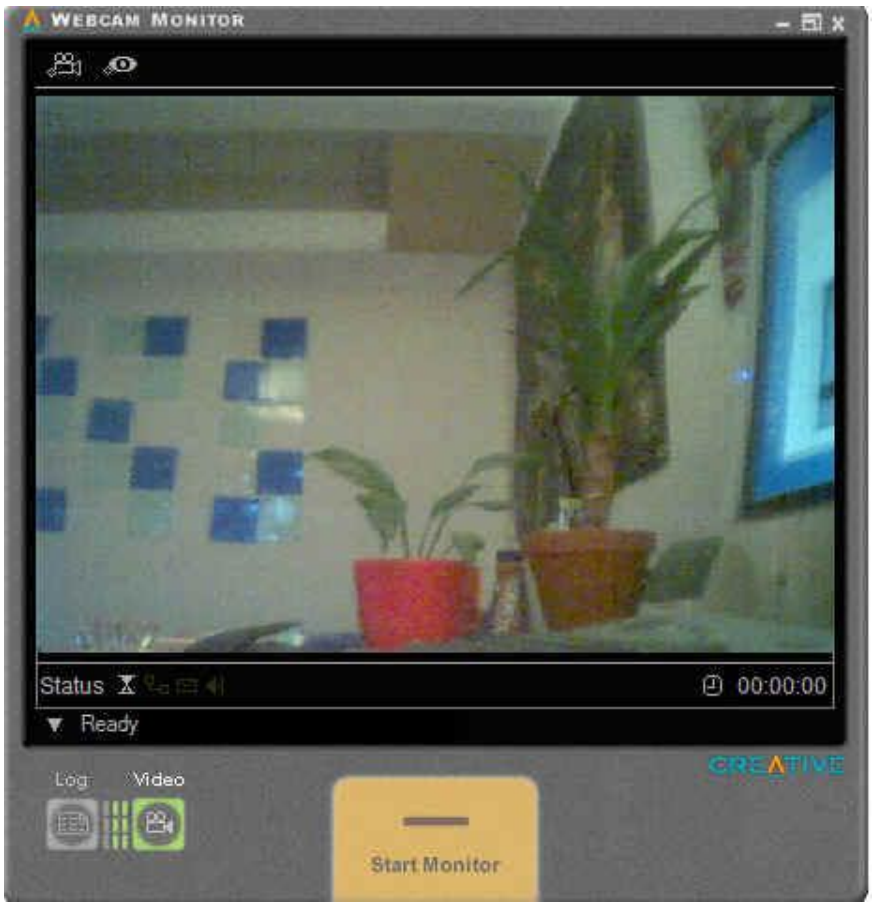

Abbildung 4: Kamerasoftware von Ceative

3 Überblick über die parallele Schnittstelle

Der Parallelport gehört mit den seriellen Schnittstellen zu den ältesten Anschlussmöglichkeiten von externer Hardware an den PC. Die parallele Schnittstelle (auch Centronics-Schnittstelle genannt) hat 25 Pins. Alle Signale besitzen TTL-Pegel, also entweder 0V oder 5V.

Der erste Parallelport (LPT1) wird zumeist unter der Basisadresse 378h, der zweite unter 278h angesprochen. Die Basisadresse wird benötigt, um die Daten-, Status- und Steuerregister der Schnittstelle zu beschreiben oder auszulesen.
Ins Datenregister können 8 Bit Daten zum Transfer an das angeschlossene Gerät geschrieben werden. Durch einen Strobe-Impuls, werden die Daten vom Gerät übernommen. Die Adresse des Datenregisters entspricht der Basisadresse.
Das Statusregister (Basisadresse +1h) kann nur gelesen werden. Datentransferrichtung vom Gerät zum PC. Die Bits werden vom ange-

schlossenen Gerät gesetzt und dienen der Statusrückmeldung an den PC. Die Bezeichnungen der Statusbits, wie z.B. PaperOut lassen sich damit erklären, dass die parallele Schnittstelle ursprünglich nur für den Anschluss von Druckern konzipiert worden ist.
Über die Basisadresse +2h kann das Steuerregister beschrieben und gelesen werden. Datentransferrichtung vom PC zum Gerät. Durch setzten des DataDirection Bits lässt sich der Druckerport auch bidirektional betreiben, was ursprünglich nicht vorgesehen war. D.h. Geräte sollten nur ihren Status an den PC zurückmelden, aber sonst keine Daten an ihn senden.

3.1 IEE1284 Parallelport Modi

Erst seit 1994 ist der Parallelport durch Einführung des IEE1284-Standards offiziell auf bidirektionalen Betrieb erweitert worden. IEE1284 beschreibt fünf verschiedene Betriebsarten für den Parallelport. Diese sind:

- Compatible Mode
- Nibble Mode
- Byte Mode
- Extended Parallel Port
- Enhanced Capability Mode

3.1.1 Compatible Mode

Der Compatible Mode (auch SPP) verhält sich wie der ursprüngliche Centronics Port, um die Kompatibilität zu älteren Geräten zu gewährleisten. Die maximale Datenübertragungsrate für diesen Modus beträgt 150 Kbyte pro Sekunde.

Nibble Mode signal name	Compatibility Mode signal name	Nibble Mode signal description
PtrClk	nAck	Reverse data transfer phase: Used to qualify data being sent to the host. Reverse idle phase: Set low then high by the peripheral to cause an interrupt indicating to the host that data is available.
PtrBusy	Busy	Reverse data transfer phase: Data bit 3 then 7, then forward channel busy status.
AckDataReq	PError	Reverse data transfer phase: Data bit 2 then 6. Reverse idle phase: Set high until the host requests a data transfer, then follows nDataAvail(nFault).
Xflag	Select	Reverse data transfer phase: Data bit 1 then 5.
HostBusy	nAutoFd	Reverse data transfer phase: Set low to indicate that host can receive peripheral-to-host data, then set high to acknowledge receipt of that nibble. Following a reverse channel transfer, the interface transitions to idle phase when HostBusy(nAutoFd) is set low and the peripheral has no data available. Reverse idle phase: Set high in response to PtrClk(nAck) low pulse to re-enter reverse data transfer phase. If set high with IEEE 1284 Active(nSelectIn) set low, the IEEE 1284 idle phase is aborted, and the interface returns to Compatibility Mode.
nDataAvail	nFault	Reverse data transfer phase: Set low to indicate that the peripheral has the data ready to send to the host. Then used to send data bit 0 (LSB) then 4. Reverse idle phase: Used to indicate that data is available.

Abbildung 5: Nibble Modus

3.1.2 Nibble Mode

Bei Verwendung des Nibble Modes werden vier Bit des Statusregisters zur Datenübertragung vom Gerät zum PC benutzt. Da nur vier Bit zur Verfügung stehen wird ein Byte in zwei Hälften übertragen. Das Datenregister wird im Nibble Mode nicht verwendet. Die maximale Datenübertragungsrate für diesen Modus beträgt 50 Kbyte pro Sekunde.

3.1.3 Byte Mode

Im Byte Mode werden die Daten vom PC zum Gerät, als auch vom Gerät zum PC über das Datenregister übertragen. Im Gegensatz zum Nibble Mode werden die Daten byteweise gesendet, bzw. empfangen. Die Übertragungsrichtung wird im Steuerregister über das DataDirection Bit festgelegt. Die maximale Datenübertragungsrate für diesen Modus beträgt 150 Kbyte pro Sekunde.

Byte Mode signal name	Compatibility Mode signal name	Byte Mode signal description
HostClk	nStrobe	Will be pulsed low during Byte Mode transfers. The peripheral shall ensure that HostClk(nStrobe) does not cause data to be latched when this happens.
Data1...Data8	Data1...Data8	Reverse channel data.
PtrClk	nAck	Reverse data transfer phase: Used to qualify data being sent to the host. Reverse idle phase: Set low then high by the peripheral to cause an interrupt indicating to the host that data is available.
PtrBusy	Busy	Forward channel busy status.
AckDataReq	PError	Reverse data transfer phase: Follows nDataAvail(nFault). Reverse idle phase: Set high until the host requests a data transfer, then follows nDataAvail(nFault).
HostBusy	nAutoFd	Reverse data transfer phase: Set low to indicate that the host can receive peripheral-to-host data, then set high to acknowledge receipt of that byte. Following a reverse channel transfer, the interface transitions to idle phase when HostBusy(nAutoFd) is set low and the peripheral has no data available. Reverse idle phase: Set high in response to PtrClk(nAck) low pulse to re-enter reverse data transfer phase. If set high with IEEE 1284 Active(nSelectIn) set low, the IEEE 1284 idle phase is aborted, and the interface returns to Compatibility Mode.
nDataAvail	nFault	Set low to indicate that the peripheral has data to send to the host.

Abbildung 6: Byte Modus

3.1.4 Extended Parallel Port

Der Extendet Parallel Port (EPP) ist abwärtskompatibel zum SPP Modus. Neben dem Daten-, Status- und Steuerregister bietet der EPP die Möglichkeit fünf weiter Register zur Datenübertragung zu nutzen. Die zusätzlichen Register befinden sich von Basisadresse +3h bis Basisadresse +7h. Die Datenübertragungsrate beträgt maximal 2 Mbyte pro Sekunde.

EPP Mode signal name	Compatibility Mode signal name	EPP Mode signal description
nWrite	nStrobe	Set low to denote an address or data write operation to the peripheral. Set high to denote an address or data read operation from the peripheral.
AD1...AD8	Data1...Data8	Host-to-peripheral or peripheral-to-host address or data.
Intr	nAck	Used by the peripheral to interrupt the host. This signal is pulsed low for a minimum of 500nS and must be raised high for a minimum of 1uS before re-asserting.
nWait	Busy	This signal should be driven inactive as a positive acknowledgment from the peripheral that transfer of data or address is completed. The signal is active low. It should be driven active as an indication that the device is ready for the next address or data transfer.
nDStrb	nAutoFd	This signal is active low. It is used to denote a data cycle.
nInit	nInit	This signal is active low. When driven active (low), this signal initiates a termination cycle that results in the interface returning to Compatibil ity Mode.
nAStrb	nSelectIn	This signal is used to denote an address cycle. It is active low.
User defined 1	PError	This signal is manufacturer specific and beyond the scope of this standard.
User defined 2	nFault	This signal is manufacturer specific and beyond the scope of this standard. It is commonly used to indicate data is available as in ECP mode.
User defined 3	Select	This signal is manufacturer specific and beyond the scope of this standard.

Abbildung 7: EPP Modus

3.1.5 Enhanced Capability Mode

Der Enhanced Capability Mode (ECP) ist ebenso wie der EPP abwärtskompatibel und bietet weitere Register zur Nutzung. Hinzu kommt noch die Fähigkeit verlustlose RLE-Komprimierung bei der Datenübertragung einzusetzen. Außerdem verfügt der ECP Modus über einen 16 k großen FIFO (First in first out) Buffer mit DMA- und Interruptfähigkeiten. Die Datenübertragungsrate beträgt ebenfalls maximal 2 Mbyte pro Sekunde.[1]

ECP Mode signal name	Compatibility Mode signal name	ECP Mode signal description
HostClk	nStrobe	Used in a closed-loop handshake with PeriphAck(Busy) to transfer data or address information from the host to the peripheral.
Data1...Data8	Data1...Data8	Host-to-peripheral or peripheral-to-host address or data.
PeriphClk	nAck	Used in a closed-loop handshake with HostAck(nAutoFd) to transfer data from the peripheral to the host.
PeriphAck	Busy	The peripheral uses this signal for flow control in the forward direction. PeriphAck also provides a ninth data bit used to determine whether command or data information is present on the data signals in the reverse direction. See section 6.9 and figure 14, events 42-46.
HostAck	nAutoFd	The host drives this signal for flow control in the reverse direction. It is used in an interlocked handshake with PeriphClk(nAck). HostAck also provides a ninth data bit that is used to determine whether command or data information is present on the data signals in the forward direction. See section 6.9 and figure 13, events 34-32.
NReverseRequest	nInit	This signal is driven low to place the channel in the reverse direction. While in the ECP Mode, the peripheral is only allowed to drive the bidirectional data signals when nReverseRequest is low and IEEE 1284 Active is high.
NPeriphRequest	nFault	During ECP Mode, the peripheral shall drive this pin low to request communications with the host. This signal provides a mechanism for peer-to-peer communication. This signal would be typically used to generate an interrupt to the host. This signal is valid in the forward and reverse directions.
NAckReverse	PError	The peripheral drives this signal low to acknowledge nReverseRequest. The host relies upon nAckReverse to determine when it is permitted to drive the data signals.

Abbildung 8: ECP Modus

[1] vgl. S.682 – 703 PC-Werkstatt

Pin	Richtung	Compatible	Nibble	Byte	ECP	EPP
1	PC→Gerät	Strobe	–	HostClk	HostClk	/Write
2	PC←→Gerät	Data 1	–	Data 1	Data 1	AD1
3	PC←→Gerät	Data 2	–	Data 2	Data 2	AD2
4	PC←→Gerät	Data 3	–	Data 3	Data 3	AD3
5	PC←→Gerät	Data 4	–	Data 4	Data 4	AD4
6	PC←→Gerät	Data 5	–	Data 5	Data 5	AD5
7	PC←→Gerät	Data 6	–	Data 6	Data 6	AD6
8	PC←→Gerät	Data 7	–	Data 7	Data 7	AD7
9	PC←→Gerät	Data 8	–	Data 8	Data 8	AD8
10	Gerät→PC	/Acknowledge	PtrClk	PtrClk	PeriphClk	Interrupt
11	Gerät→PC	Busy	Data 3, 7	PtrBusy	PeriphAck	/Wait
12	Gerät→PC	Paper Error	Data 2, 6	AckDataReq	/AckRevers	UserDefin1
13	Gerät→PC	Select	Data 1, 5	–	Xflag	UserDefin3
14	PC→Gerät	/AutoFeed	HostBusy	HostBusy	HostAck	/DataStrb
15	nicht definiert					
16	Logic GND					
17	Chassis GND					
18	Peripheral Logic High					
19	GND Strobe					
20	GND Data 1					
21	GND Data 2					
22	GND Data 3					
23	GND Data 4					
24	GND Data 5					
25	GND Data 6					
26	GND Data 7					
27	GND Data 8					
28	GND Paper Error, Select, Acknowledge					
29	GND Busy, Fault					
30	GND Auto Feed, Select In, Init					
31	PC→Gerät	/Init	–	–	/RevRequest	/Reset
32	Gerät→PC	/Fault	Data (0, 4)	/DataAvail	/PeriRequest	UserDefin 2
33	nicht definiert					
34	nicht definiert					
35	nicht definiert					
36	PC→Gerät	/SelectIn	1284Active	1284Active	1284Active	/AddressStrb

Abbildung 9: IEEE1284 Modi

3.2 Auswahl des Übertragungsmodus

Für die Ansteuerung der Creative Video Blaster II Webcam in Verbindung mit einem Controller kommen theoretisch alle fünf Übertragungsmodi in Betracht. Allerdings ist die Ansteuerung im vollen ECP Modus durch DMA und Interrupts sehr aufwendig. Welcher Modus zum Einsatz kommt ist in erster Linie davon abhängig, welche Modi Kamera und Controller unterstützen. Zumindest für die Kamera ist im Handbuch angegeben, dass der Standard Drucker Port (also SPP), 8-Bit bidirektional (Byte oder EPP) und ECP Enhanced Modus unterstützt werden. Ob die Kamera die vollen 2 Mbyte/s im ECP Modus zur Datenübertragung nutzen kann bleibt vorerst ungeklärt. Nach Ver-

suchen mit der zur Kamera mitgelieferten Software Webcam Monitor, liegt die Vermutung nahe, dass der ECP Modus in einer abwärtskompatiblen Einstellung betrieben wird, da die durchschnittliche Übertragungsgeschwindigkeit bei ca. 150 Kbytes pro Sekunde zu liegen scheint.

3.3 Zugriff auf den Parallelport unter Windows

Die erste große Hürde für eine erfolgreiche Kommunikation mit der Kamera ist die Ansteuerung der Parallelports. In den meisten Programmiersprachen gibt es Befehle zum Auslesen und Schreiben von Daten am Druckerport, z.B. _inp() oder _outp(). Diese Lese- und Schreibzugriffe funktionieren unter älteren DOS-Systemen, Windows95 und Windows98 ohne Probleme. Mit der Einführung von WindowsNT hat sich dies jedoch geändert.

„Being a very secure operating system, Windows NT assigns some privileges and restrictions to different types of programs running on it. It classifies all the programs in to two categories, User mode and Kernel mode ie; running in ring3 and ring0 modes. User mode programs are running in ring3 mode and Kernel mode programs are running in ring0 mode. The programs you generally write falls in the user mode category. The user mode programs are restricted to use certain instructions like IN, OUT etc.. Whenever the operating system finds that a user mode program is trying to execute such instructions, the operating system stops execution of those programs and will display an error message.” [1]

Mit anderen Worten: WindowsNT, Windows2000 und WindowsXP verbieten Zugriffe auf den Parallelport im User-Mode. Beim Versuch auf den Druckerport mit einem selbstgeschriebenen Programm zuzugreifen, erscheint eine Fehlermeldung.

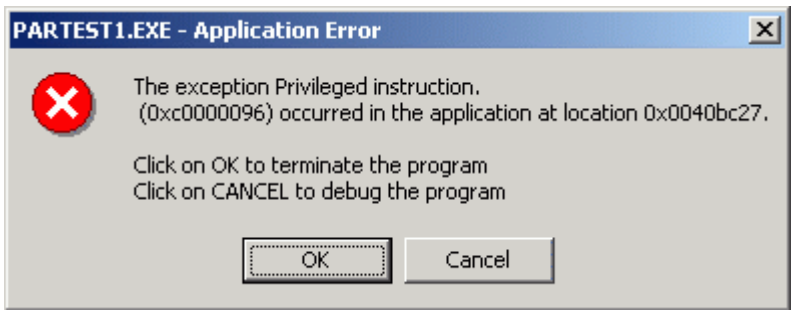
PARTEST1.EXE - Application Error
The exception Privileged instruction.
(0xc0000096) occurred in the application at location 0x0040bc27.
Click on OK to terminate the program
Click on CANCEL to debug the program
OK
Cancel

Abbildung 10: Fehlermeldung unter Windows XP

[1] http://www.logix4u.net/inpout32.htm

„Kernel mode programs are in no way restricted in executing such instructions. Device drivers are capable of running in kernel mode. So the workaround for the above stated problem is to write a kernel mode driver capable of reading and writing data to parallel port and let the user mode program to communicate with it."[1]

Die Lösung des Problems liegt also in der Benutzung eines Programms, das im Kernel-Mode ausgeführt wird und somit vollen Zugriff auf den Parallelport hat. Dies kann in Form eines Gerätetreibers geschehen, der dem Benutzer eine Schnittstelle zu Kommunikation mit dem Parallelport anbietet.

3.4 Portzugriffe mit Java

Ursprünglich war geplant, die Kameraansteuerung in Java zu programmieren. Java besitzt von Haus aus eine API, die Java Communications API[2]., mit der man den Parallelport ansteuern kann.
Unter WindowsXP bleibt der Zugriff auf den Druckerport mit der Java Communications API aber aus den oben genannten Gründen verwehrt. Eine Zugriffsmöglichkeit unter Java bietet Parport von Juan Gabriel Del Cid Portillo[3]. Nach dem Herunterladen des zip-Archivs und Befolgen der Installationsanleitung gelingt der Zugriff auf den Druckerport.
Alternativen sind hier RxTx[4] und Java Comm for Linux (JCL)[5]. Speziell Java Comm for Linux und RxTx sind auf Parallelportzugriffe unter Linux ausgelegt. Allerdings wurden diese Alternativen hier nicht getestet und es kann somit keine verbindliche Aussage getroffen werden, ob die Benutzung des Parallelports damit wirklich funktioniert.

Nachdem die Ansteuerung der Druckerschnittstelle unter Java gelang, wurde dennoch die Programmiersprache gewechselt und auf C/C++ umgestiegen. Dadurch bedingt erfolgte die Programmierung nicht mehr mit Eclipse, sondern mit DevC++ und VisualStudio. Die Gründe hierfür waren keine Mängel der Programmiersprache Java, bzw. Vorteile bei C/C++, sondern rein praktischer Natur. Unter SourceForge[6] fand sich ein Projekt mit DOS-Quellcode der Kamerasoftware vom Hersteller Creative unter der GNU Public License. Um diesen testen zu können, war der Umstieg auf C/C++ sinnvoll.

[1] http://www.logix4u.net/inpout32.htm
[2] http://java.sun.com/products/javacomm/
[3] http://www.geocities.com/Juanga69/parport
[4] http://www.rxtx.org/
[5] http://www.geeksville.com/~kevinh/linuxcomm.html
[6] http://webcam.sourceforge.net

3.5 Portzugriffe mit C/C++

Auch die Portzugriffe unter C/C++ unterliegen den Restriktionen des eingesetzten Betriebssystemes WindowsXP. Für den Zugriff auf den Druckerport kommt hier inpout32[1] zum Einsatz. Die Portzugriffe funktionieren damit problemlos und unkompliziert. Auch lässt sich der Code ohne Probleme in DevC++ und VisualStudio einbinden.

3.6 Installation und Benutzung von Inpout32.dll

Zur Benutzung von inpout32 muss man die Datei inpout32.dll ins Windows-Systemverzeichnis oder in den Order mit dem ausführbaren Quellcode kopieren. Im Quellcode selbst muss man die Libary inpout32.dll initialisieren. Das könnte z.B. so aussehen:

```
/* prototype (function typedef) for DLL function Inp32: */

    typedef short _stdcall (*inpfuncPtr)(short portaddr);
    typedef void _stdcall (*oupfuncPtr)(short portaddr, short datum);

int initialiseInp32(void){
    HINSTANCE hLib;
    inpfuncPtr inp32;
    oupfuncPtr oup32;

    /* Load the library */
    hLib = LoadLibrary("inpout32.dll");
    printf("inpout32.dll loaded\n");

    if (hLib == NULL) {
        printf("LoadLibrary Failed.\n");
        return -1;
    }

    /* get the address of the function */

    inp32 = (inpfuncPtr) GetProcAddress(hLib, "Inp32");

    if (inp32 == NULL) {
        printf("GetProcAddress for Inp32 Failed.\n");
        return -1;
    }

    oup32 = (oupfuncPtr) GetProcAddress(hLib, "Out32");

    if (oup32 == NULL) {
        printf("GetProcAddress for Oup32 Failed.\n");
        return -1;
    }
    printf("inpout32.dll ready for use  \n");
}
```

Abbildung 11: Initialisierung von inpout32.dll

[1] http://www.logix4u.net/inpout32.htm

Nun kann über (outp32)(portAdresse, wert) ein Wert auf den unter portAdresse angegebenen Parallelport geschrieben werden. (inp32)(portAdresse) liest den Port aus und liefert einen Short-Wert der aktuellen Belegung zurück. Portadresse und Wert sind auch jeweils Short-Werte.

Folgende Grafik zeigt den internen Ablauf bei Aufrufen von inp32 oder outp32:

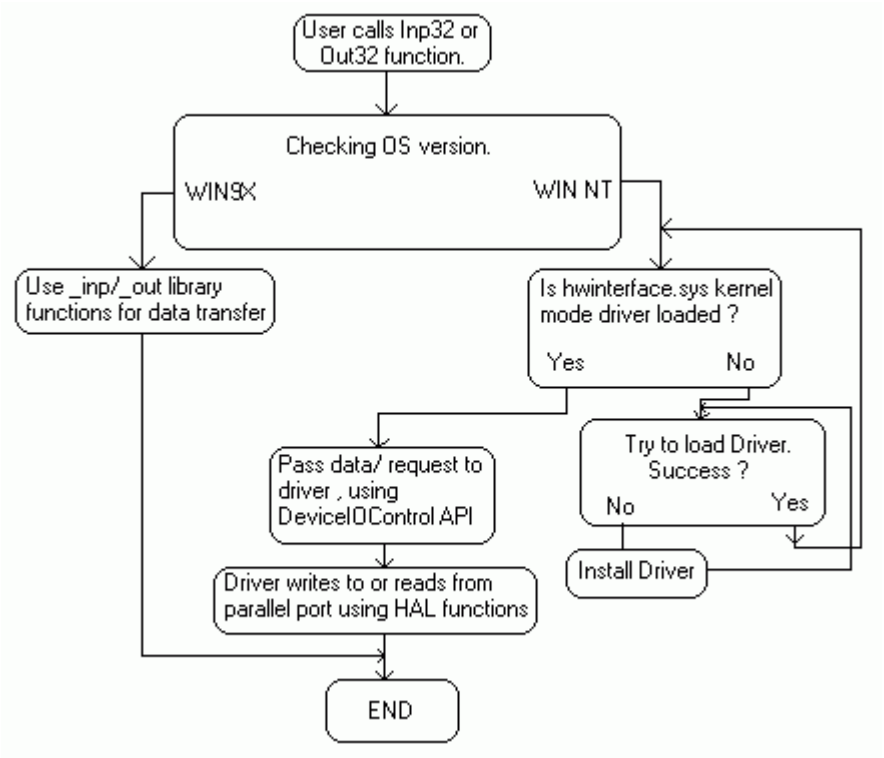

Abbildung 12: Ablaufdiagramm von inpout32.dll

Am Ende des Programms kann die inpout32.dll wie folgt entladen werden:

```
int deinitialiseInp32(void)
{
    FreeLibrary(hLib);
    return 0;
}
```

Abbildung 13: Deinitialisierung der inpout32.dll

Eine Alternative zur inpout32.dll bietet ParallelPort[1] von PJ Naughter. Die Vorgehensweise zur Benutzung ist sehr ähnlich.

[1] http://www.naughter.com

3.7 Kontrollwerkzeug für Parallelportzugriffe

Als sehr nützlich hat sich das Programm Parallel Port Monitor[1] von Neil Fraser erwiesen. Das Programm überwacht einen gewählten Parallelport und zeigt dessen aktuelle Belegung des Daten-, Status- und Steuerregisters an. Die Software arbeitet im Hintergrund und erlaubt anderen Programmen weiterhin den Zugriff auf den Port. Ob eine Datenübertragung stattfindet lässt sich somit leicht kontrollieren. Ferner bietet das Programm auch die Möglichkeit die Bits im Daten- und im Steuerregister per Mausklick zu manipulieren.

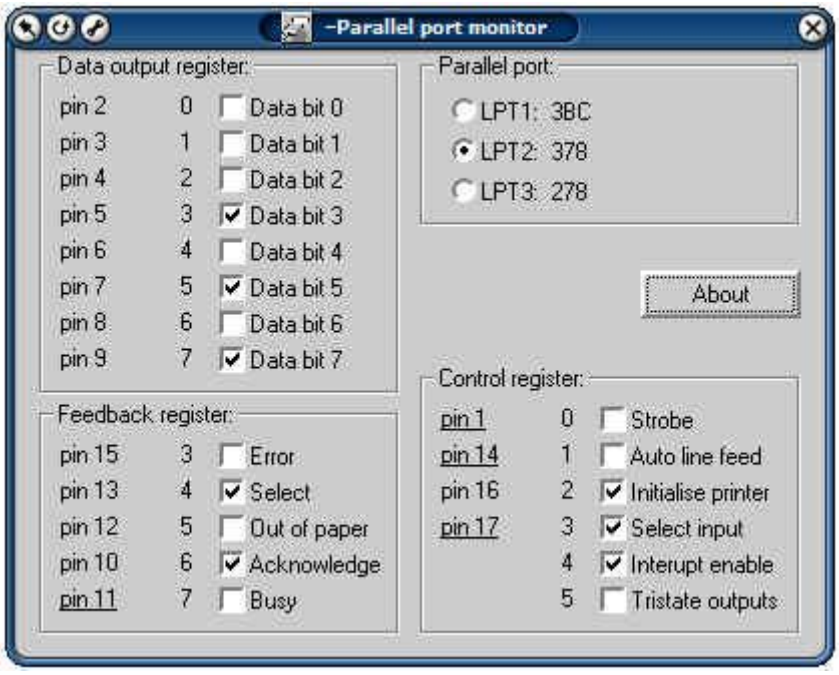
-Parallel port monitor
Data output register:
pin 2 0 Data bit 0
pin 3 1 Data bit 1
pin 4 2 Data bit 2
pin 5 3 Data bit 3
pin 6 4 Data bit 4
pin 7 5 Data bit 5
pin 8 6 Data bit 6
pin 9 7 Data bit 7
Feedback register:
pin 15 3 Error
pin 13 4 Select
pin 12 5 Out of paper
pin 10 6 Acknowledge
pin 11 7 Busy
Parallel port:
LPT1: 3BC
LPT2: 378
LPT3: 278
About
Control register:
pin 1 0 Strobe
pin 14 1 Auto line feed
pin 16 2 Initialise printer
pin 17 3 Select input
4 Interupt enable
5 Tristate outputs

Abbildung 14: Parallel port monitor

4 Der CPiA Prozessor

Das Herzstück der benutzten Videoblaster Webcam II von Creative bildet der CPiA (Color Processor Interface ASIC) Chip. Hier findet die Bildvorverarbeitung und die Kommunikation mit der Außenwelt statt. Der CPiA Chip besteht aus drei Modulen. Einem Videoprozessor (VP), einem Videokompressor (VC) und einem Kontrollprozessor (CP).

[1] http://neil.fraser.name/software/ltp/

4.1 Videoprozessor

Der in der Kamera verwendete VVL404 CMOS Bildsensor liefert die reinen Bilddaten an den Videoprozessor. Dieser überführt die Daten in ein 4:2:2, bzw. 4:2:0 YUV-Format (in YUYV Reihenfolge). Die Bildgröße beträgt 352*288 (CIF) Pixel oder 176*144 (QuarterCIF, QCIF) Pixel. Anschließend stehen die Bilddaten dem Videokompressionsmodul zur Verfügung.

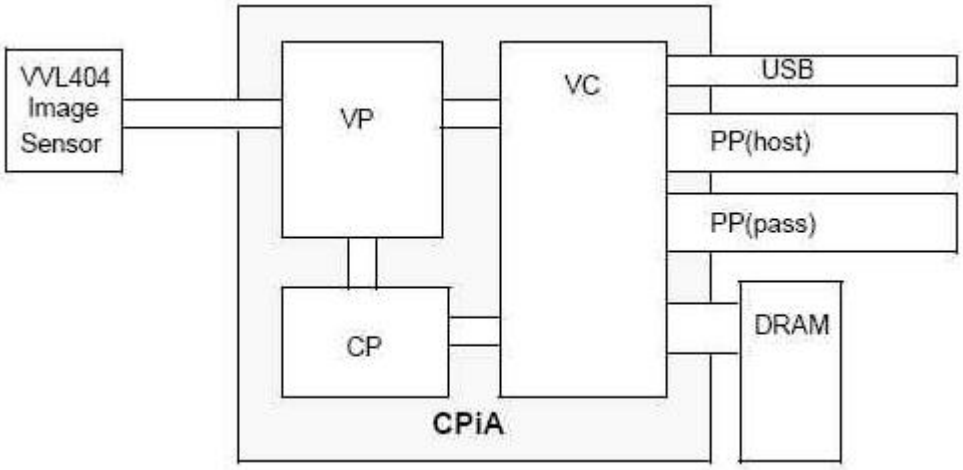

Abbildung 15: CPiA System Block Diagramm

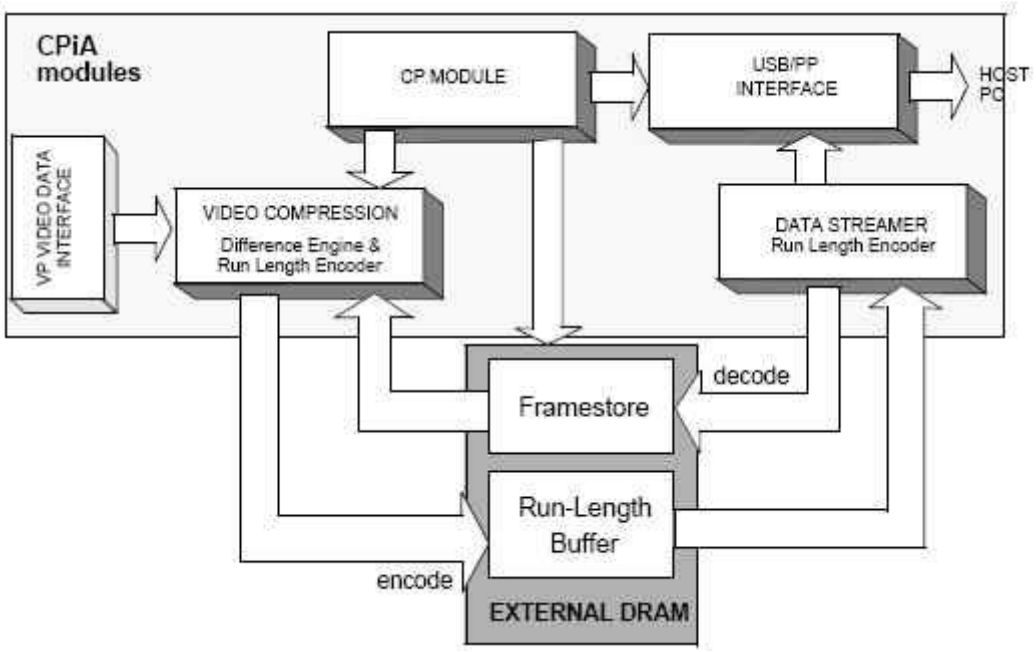

Abbildung 16: Die CPiA Bilddatenverarbeitung

4.2 Videokompressionsmodul

Das Videokompressionsmodul bietet die Möglichkeit die Bilddaten unbearbeitet zu lassen oder auf zwei Arten zu komprimieren. Die erste Möglichkeit ist eine einfache Lauflängenkodierung (RLE). Dabei werden längere Folgen von gleichen Bildwerten durch zusammenfassen gekürzt. Die zweite Möglichkeit ist eine Differenzkodierung. Bei der Differenzkodierung wird zu jedem Pixel eines neuen Bildes ein Vergleich zum vorherigen Bild durchgeführt und nur noch die Bildwerte übertragen, die sich geändert haben. Die Lauflängenkodierung ist dann sehr effektiv, wenn es viele Bereiche mit gleichen Farben und Helligkeiten im Bild gibt. Im Allgemeinen ist die Differenzkodierung wesentlich effizienter. Vorraussetzung dafür ist aber, dass sich der Bildinhalt möglichst wenig verändert. Für die Erhöhung der Kompression benutzen beide Verfahren einen Schwellwert, der festlegt mit welcher Toleranz Bildwerte als gleich anzusehen sind. Beide Verfahren sind dann verlustlos, wenn der Schwellwert so eingestellt ist, dass nur wirklich identische Werte als gleich gelten. In der Praxis macht dies bei der verwendeten Webcam jedoch keinen Sinn und erhöht nur das zu übertragende Datenvolumen. Die beschrieben Verfahren können beide gleichzeitig eingesetzt werden. Der CPiA Prozessor bietet zudem die Möglichkeit die Kompression auf Automatik zu setzen.

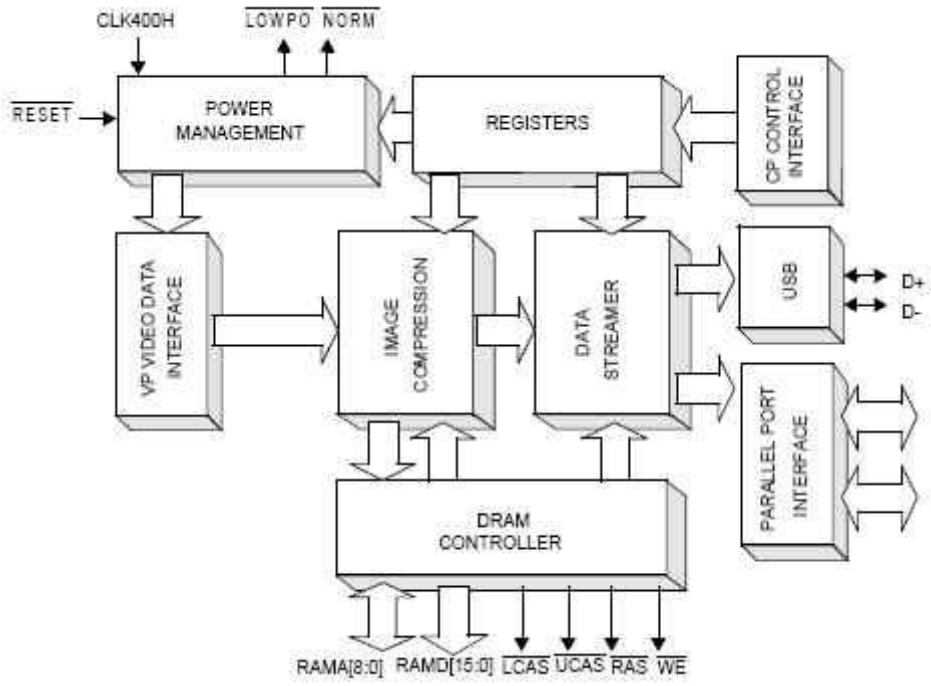

Abbildung 17: Das CPiA Videokompressionsmodul

4.3 Kontrollprozessor

Der Kontrollprozessor steuert den Videoprozessor und Videokompressor. Er kontrolliert die automatische Blende/Belichtung und die Farbbalance der Videobilder. Zudem übernimmt er die Kommunikation mit dem PC über parallele Schnittstelle. Diese hält sich an Vorgaben der IEEE1284 für die parallele Schnittstelle. Für an die Kamera gerichtete Nachrichten wird immer der ECP Modus verwendet. Für Datentransport von der Kamera zum PC benutzt der Kontrollprozessor entweder ECP oder den Nibble Modus, falls der Port als nicht bidirektional erkannt wird. Zudem werden auch im ECP Modus nur das Daten-, Status- und Steuerregister verwendet.

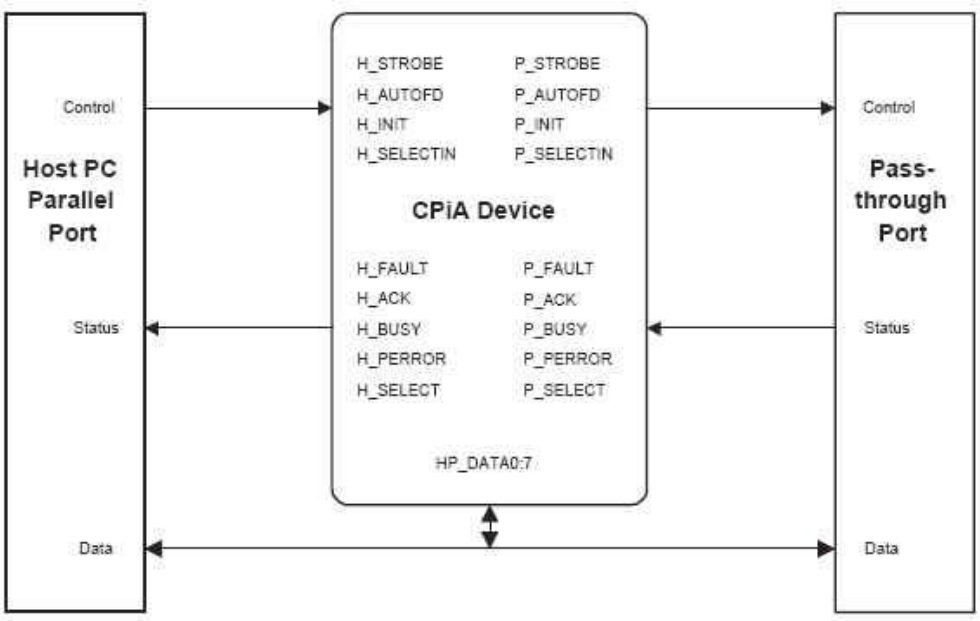

Abbildung 18: Benutzte Parallelport-Pins

"Data is transfered in 8 byte packets, each packet being preceeded by the 1284 negotiation and setup sequences, and terminated with a 1284 termination sequence. Commands are passed to CPiA in a single 8 byte packet, termed a command transfer. This can optionally be followed by the transfer of a data transfer. Bytes within the command packet are used to specify whether a data packet is to follow, and its direction. Most commands are not accompanied by data packets."[1]

Die Kommandos zur Steuerung des CPiA Prozessors und somit der Kamera sind im Dokument „Software Developer's Guide for CPiA Cameras" näher beschrieben.

[1] Vision CPiA Data Sheet.pdf Kap. 2.2.5.8 Command / Status Transfers

5 Ansteuerung der Kamera

5.1 Vorhandener Quellcode

Unter Sourceforge[1] findet sich ein Projekt, das sich ebenfalls der Ansteuerung von Webcams mit CPiA Controller widmet. Im Rahmen der GNU Public License sind dort Quellcodedateien von STMicroelectronics veröffentlicht.
Windows ppc2.tar.gz enthält rund 65 Dateien und pp-test.zip 10 Dateien in der Programmiersprache C. Letzteres ist ein Programm zur einfachen Funktionsprüfung von CPiA Webcams. Da der Code aus dem Jahr 1999 stammt und zur Ausführung unter DOS gedacht ist, lässt er sich nicht ohne aufwendige Modifikationen kompilieren. Zusätzlich sind nicht alle inkludierten Dateien im Archiv enthalten. Diese sind jedoch teilweise im anderen Archiv enthalten. Aus dem Bestreben den Quellcode wieder lauffähig zu machen, ist eine erste Programmversion zur Ansteuerung der Creative VideoBlaster Webcam entstanden.

5.2 PP_test

Dieses Programm bietet die Möglichkeit die grundlegenden Kamerafunktionen zu testen. Neben der Einstellung verschiedener Kameraparameter, können Bilder aufgenommen und auf der Festplatte im PPM-Format abgespeichert werden. Diese Version ist auch die Grundlage für spätere Programmversionen.
Der Quellcode umfasst 17 C-Dateien mit entsprechenden Headerdateien. Die Dateien lassen sich nach ihrer Funktionalität in die vier Gruppen Datenaustausch, Befehlsumsetzung, Konvertierung und Steuerung aufteilen.

5.2.1 pport.c, comm.c und cpialink.c

Für den Datenaustausch mit der Webcam über die parallele Schnittstelle sind die Dateien pport.c, comm.c und cpialink.c verantwortlich. Pport.c definiert die Pins des Druckerports und die Datenübertragungsmodi Nibble und ECP. Mit den entsprechenden Methoden können die vorhanden Druckerports detektiert und initialisiert werden, so dass sie dem Programm zur Verfügung stehen. Je nach erkanntem Port, stehen verschieden Übertragungsmodi für den Datentransfer bereit. Neben ECP und Nibble gibt es noch einen weiteren Modus Namens SimECP. Dieser Modus ist schneller als eine reine Nibble-Übertragung und fast so schnell wie der normale ECP-Modus. Für die Übertragung stehen für jeden Modus Schreib- und Lesemethoden bereit, z.B. *ECPWriteBuffer( ), ECPReadBuffer( ).* Über die Methode *setTransferMode( )* kann der Übertragungsmodus auch manuell ausgewählt werden. Generell werden alle Schreib- oder Lesezugriffe auf

[1] http://webcam.sourceforge.net

die Kamera über pport.c ausgeführt. Da die vorhandenen Methoden *inp( )* und *outp( )* unter WindowsXP nicht auf den Druckerport zugreifen dürfen, werden sie auf *inp_32( )* und *outp*_32() umgeleitet[1]. Über comm.c ist die inpout32.dll für den Portzugriff eingebunden. In comm.c ist die Initialisierungsmethode *COMM_Init(bool test, bool writeToFile)* um zwei Parameter erweitert. Mit dem ersten bool-Wert wird beim Initialisieren explizit ein Lese- und Schreibtest auf den Parallelport ausgeführt. Mit dem zweiten Parameter kann sämtlicher Datentransport über *inp_32( )*, bzw. *outp_32( )* in eine Textdatei protokolliert werden.
Auf pport.c baut cpialink.c auf. Auch hier gibt Methoden zur Initialisierung und Portdetektion. Diese verwenden u.a. die von pport.c bereitgestellten Funktionen. Die wichtigsten Methoden von cpialink.c sind *CLINK_TransferMsg( )* und *CLINK_UploadStreamData( )*. *CLINK_TransferMsg( )* erhält als Eingaben die Anzahl der zu übertragenden Bytes und einen Verweis auf die Daten. Diese werden je nach erkanntem Parallelport automatisch per ECP, SimECP oder im Nibble-Modus unter Rückgriff der Methoden aus pport.c an die Kamera übertragen. Um Bilddaten von der Kamera zu lesen, wird *CLINK_UploadStreamData( )* benutzt. Durch den Aufruf wird der von CPiA-Chip produzierte Datenstrom eines Bildes gelesen. Der Aufruf darf erst nach Aufnehmen einens Bildes erfolgen, wenn die Kamera im Zustand *STREAM_READY* ist. Als Parameter erhält diese Methode ebenfalls Verweise auf einen Buffer zur Datenablage, Bytezahl und die aktuell gelesenen Bytes.

5.2.2 cmd.h und cpia_cmd.c

Cmd.h und cpia_cmd.c enthalten die Definitionen und Implementierung aller Steuerbefehle in entsprechenden Methoden für CPiA-Controller. D.h., Cpia_command.c umfasst alle Befehle die im Software Developer's Guide for CPiA Cameras beschrieben sind. Die Befehle dienen der Einstellung aller Parameter für den Videoprozessor (z.B. Belichtung, Farbparameter), den Kontrollprozessor (z.B. HiPower, BaudRate, GrabMode) sowie das Videokompressionsmodul (z.B. Kompressionsrate). Bei Aufruf einer der Methoden aus cpia_cmd.c mit passenden Parametern, wird der entsprechende Befehl erstellt und über *CLINK_TransferMsg( )* an die Kamera geleitet. Je nach Befehl werden Antworten der Kamera ebenfalls ausgelesen und stehen dann zur Auswertung zu Verfügung.

5.2.3 cconvert.c, decode.c und cpia_vid.h

Cconvert.c dient der Umwandlung der von der Kamera gelieferten Bilddaten. Grundsätzlich liefert die Kamera die Bilddaten im YUV oder UVUY Format mit einer Abtastung von 4:2:2 oder 4:2:0. Die in

[1] Siehe Kapitel 3 ff

Cconvert.c enthaltenen Methoden erlauben es die Bildinformationen von YUV nach RGB mit 16-, 24- oder 32-Bit Farbtiefe zu konvertieren. Als Eingabeparameter werden Verweise auf einen Buffer mit den vorhandenen YUV-Werten und auf einen weitern Buffer für die RGB-Werte benötigt.

```
void YUVPixeltoRGB24(unsigned char  *YUVPixel, unsigned char  *RGBPixel )
{
   int yval, uval, vval, yval2;
   int guv, rv, bu;

   yval = *YUVPixel++ - 16;
   uval = *YUVPixel++;
   yval2 = *YUVPixel++ - 16;
   vval = *YUVPixel++;

   uval -= 128;
   vval -= 128;
   yval = 19 * yval;
   yval2 = 19 * yval2;

   bu =  (uval<<5);
   guv =  (- 6*uval - 13*vval);
   rv =  26*vval;

   *RGBPixel++ = (unsigned char)limit(((yval + bu)>>4) );
   *RGBPixel++ = (unsigned char)limit(((yval + guv )>>4));
   *RGBPixel++ = (unsigned char)limit(((yval + rv)>>4)  );

   *RGBPixel++ = (unsigned char)limit(((yval2 + bu)>>4) );
   *RGBPixel++ = (unsigned char)limit(((yval2 + guv )>>4));
   *RGBPixel++ = (unsigned char)limit(((yval2 + rv)>>4)  );
   return;
}
```

Abbildung 19: YUVPixeltoRGB24( )

Decode.c zeichnet sich verantwortlich für die Dekodierung der gelieferten Bilddaten. Eine Dekodierung ist nötig, sobald die Daten komprimiert von der Kamera übertragen werden oder wenn die Daten in ein anderes Farbformat konvertiert werden sollen. Neben der Konvertierung in RGB ist es mit den Methoden von decode.c auch möglich folgende Formate zu erhalten: I420, YV12 (4:2:0, planar), UYVY (4:2:2) und YUY (4:2:2).
Nach der Initialisierung mit *InitDecode( )* ist es mit *VerifyStream( )* möglich, die von der Kamera gelieferten Bilddaten eines Frames auf ihre Korrektheit hin zu überprüfen. Dieser Schritt ist jedoch optional, da die Bilddaten sehr zuverlässig und immer korrekt von der Kamera übertragen werden. Bei Nutzung von *VerifyStream( )* kann es unter Verwendung des QCIF-Bildformates und aktivierter Kompression zu Fehlermeldungen kommen. Die gelieferten Bilddaten sind aber den-

noch korrekt und lassen sich ohne Probleme weiter verwenden. Um die Bilder später in einem einfachen PPM-Bildformat speichern zu können, werden die Bilddaten in ein RGB-Format mit 24-Bit Farbtiefe umgewandelt. Der gewünschte Farbraum wird mit *FormatToSupply-FourCC( )* festgelegt. Wichtigster Schritt ist der anschließende Aufruf von *FormatStream()*. Als Eingabeparameter benötigt *FormatStream()* ein gültiges Farbformat, Verweise auf den Buffer mit den Bilddaten und einen weiteren Buffer in dem das Frame im neuen Format gespeichert werden soll. Zusätzlich werden noch die Bildgröße und eine Information über die Kompression benötigt. Diese Angaben lassen sich auch ohne Wissen über die zur Bildaufnahme verwendeten Einstellungen aus jedem Frameheader auslesen. *FormatStream()* arbeitet sehr effektiv, da Dekompression und Farbraumkonvertierung in einem Schritt ausgeführt werden. Bei aktivierter Differenzkodierung werden nur die Pixel übertragen, die sich um einen gewissen Betrag in ihrem Farbwert verändert haben. Nur diese Pixel konvertiert *FormatStream()* in den gewünschten Farbraum und überschreibt die betroffenen Farbwerte im Ausgabebuffer. Zuerst ein vollständiges Bild zu erstellen und dieses dann komplett in einen anderen Farbraum zu transformieren, wäre ineffizienter. Diese Vorgehensweise tritt aber genau dann ein, wenn keine Kompression verwendet wird. Sowohl die Datenübertragung als auch die Farbraumkonvertierung dauern in diesem Fall wesentlich länger. Es ist also in jedem Fall empfehlenswert die Kompression zu aktivieren. Zudem sollte nicht unerwähnt bleiben, dass die Konvertierung von YUV nach RGB recht teuer ist und daher nur durchgeführt werden sollte, wenn es wirklich notwendig ist.[1]

```
COLOUR_FORMAT  FormatToSupplyFourCC(
   unsigned long FourCC,          /* == FourCC code for output format == */
   unsigned short BitDepth)       /* == bit depth for rgb fourcc's == */{

COLOUR_FORMAT colourformat = NOT_SUPPORTED;

   switch (FourCC){
      case FOURCC_BI_RGB:
      case FOURCC_RGB:    switch (BitDepth){
                          case 8:       colourformat = NOT_SUPPORTED;
                                        break;
                          case 16:      colourformat = RGB16;
                                        break;
                          case 24:      colourformat = RGB24;
                                        break;
                          case 32:      colourformat = RGB32;
                                        break;
                          default:      colourformat = NOT_SUPPORTED;
                                        break;
                     }
```

[1] Vision CPiA Data Sheet.pdf Kap. 3.8 Image Reformating / Colour Space Conversion

```
                break;
    case FOURCC_IYUV:
    case FOURCC_I420:   colourformat = I420;
                break;
    case FOURCC_YV12:   colourformat = YV12;
                break;
    case FOURCC_YVU9:   colourformat = NOT_SUPPORTED;
                break;
    case FOURCC_UYVY:   colourformat = UYVY;
                break;
    case FOURCC_YUY2:   colourformat = YUY2;
                break;
    default:        colourformat = NOT_SUPPORTED;
                break;
  }
  return colourformat;
}
```

Abbildung 20: Format ToSupplyFourCC( )

Für die Verwendung von *FormatStream()* ist es in jedem Fall wichtig, die Ausgabebuffer in der richtigen Größe zu wählen, da *FormatStream( )* sonst abbricht. Der Ausgabebuffer sollte beim Anlegen mit Farbwerten, z.B. in blau oder schwarz initialisiert werden. Falls die Kamera direkt Bilder mit aktivierter Kompression liefert soll, ohne das vorher Vollbilder ohne Kompression übertragen wurden, dann berechnet der CPiA-Chip die Differenz zum letzten Bild im Speicher der Kamera. Nach einer Trennung vom Netz sind die Werte im Speicher nicht definiert. Es werden in jedem Fall nur Teile des ganzen Bildes übertragen. Durch das Initialisieren des Ausgabebuffers bleiben die nicht übertragen Bereiche zwar in der Farbe der Initialisierung, aber der Ausgabebuffer enthält für die Weiterverarbeitung zumindest gültige Werte. Sobald sich der Bildinhalt an den bislang noch nicht übertragenen Stellen verändert, verschwinden auch automatisch die Bildartefakte. Eine mögliche Lösung zur Vermeidung dieser Bildartefakte ist es, zuerst ein Bild ohne Kompression aufzunehmen und dann erst die Kompression einzuschalten. Dadurch wird die Entstehung von Bildartefakten direkt verhindert.

Auflösung	YUV 4:2:2	YUV 4:2:0
CIF	203 kB	153 kB
QCIF	51 kB	38 kB

Abbildung 21: Bildgrößen ohne Kompression

Die Datei cpia_vid.h enthält die Definition des Frameheaders gemäß den Vorgaben des *Software Developer's Guide for CPiA Cameras*. Neben den reinen YUV Farbinformationen wird zu jedem Bild ein Frameheader in den Datenstrom eingebaut. Der Frameheader besteht

aus den ersten 64 Bit des Datenstroms. Er startet mit den Magic Numbers 0x19 und 0x68 zur Identifikation. Des weiteren enthält er detaillierte Informationen über das Bildformat, die Kompression, dem verwendeten Bildausschnitt (ROI = region of interest) und dem Kamerastatus.

Address	Name	Description
0	HEADER0	0x19 - Magic number
1	HEADER1	0x68 - Magic number
2	TIMESTAMP_0	Time at which image was captured. Format to be decided. (Not related to the USB SOF timestamp).....
3	TIMESTAMP_1	
...		
6	TIMESTAMP_2	Time at which image was captured. Format to be decided. (Not related to the USB SOF timestamp).....
7	TIMESTAMP_3	
...		
16	VIDEO_SIZE	0 = QCIF, 1 = CIF size video
17	SUB_SAMPLE	1 = 4:2:2, 0 = 4:2:0 colour subsampling
18	YUV_ORDER	0 = YUYV, 1 = UYVY byte order
....		
24	COL_START	Capture ROI left edge
25	COL_END	Capture ROI right edge
26	ROW_START	Capture ROI top edge
27	ROW_END	Capture ROI bottom edge
28	COMP_ENABLE	0 = Non compressed image, 1 = Compressed image
29	DECIMATION	0 = No decimation, 1 = Decimation enabled.
30	Y_THRESH	Y Threshold used during compression.
31	UV_THRESH	UV Threshold used during compression
32	SYSTEM_STATE	Copy of field in CameraStatus
33	GRAB_STATE	Copy of field in CameraStatus
34	STREAM_STATE	Copy of field in CameraStatus
35	FATAL_ERROR	Copy of field in CameraStatus
36	CMD_ERROR	Copy of field in CameraStatus
37	DEBUG_FLAGS	Copy of field in CameraStatus
38	CAMERA_STATE_7	Copy of field in CameraStatus
39	CAMERA_STATE_8	Copy of field in CameraStatus
40	CR_ACHIEVED	Compression ration achieved for frame
41	FR_ACHIEVED	Framerate achieved between consecutive upload requests
63	LAST_BYTE	

Abbildung 22: Frameheader

5.2.4 main_menu.c

Main_menu.c ist die Steuerzentrale des Programms. Die main-Methode enthält die Initialisierungsaufrufe für die benötigten Komponenten und Variablen. Nach Finden eines Druckerports, wird in einer Dauerschleife das Benutzermenu dargestellt. Sobald der User eine Option des Menüs auswählt, wird der entsprechende Code ausgeführt und es erscheint wieder das Benutzermenü. Neben der Einstellung der Aufnahmeparameter (Resolution und Compression) sind die wichtigsten Menüpunkte Single Grab und Continuous Grab.

Single Grab nimmt genau ein Bild auf und speichert es als ppm-Datei auf der Festplatte. Hierbei wird die Kamera zunächst in den HiPower-Modus versetzt und die aktuellen Parameter für Bildgröße und Kompression an den CPiA-Chip der Kamera gesendet. Danach wird die Methode *SingleGrab( )* aufgerufen. Als Parameter werden Variablen zum Zählen der übertragenen Bytes, zum Speichern des Streamstates und ein bool-Wert zur Fehleranzeige übergeben. Innerhalb von *SingleGrab( )* wird als erstes *CPIA_GrabFrame()* aufgerufen. Dieser CPiA-Befehl wird über *CLINK_TransferMsg()* direkt an die Kamera gesendet und das aktuelle Bild mit den zuletzt übertragenen Aufnahmeparametern im Speicher der Kamera abgelegt. Diese Vorverarbeitung des Bildes dauert einige Millisekunden. Sobald die Bilddaten zur Übertragung bereit sind, signalisiert die Kamera dies über ein Setzen von STREAM_READY. Über die while-Schleife wird der Status der Kamera solange überprüft bis dieses Ereignis Eintritt. Sobald die Webcam bereit ist, wird die Datenübertragung per *CPiA_UploadStreamData( )* gestartet. Genauer beschrieben: *CPiA_UploadStreamData( )* ruft *CLINK_UploadStreamData( )* zur Datenübertragung auf. Die übertragenen Bilddaten werden in den bereitgestellten Buffer geschrieben.

```
int SingleGrab(unsigned long actual_bytes,unsigned char streamState, bool rtn){
//Capture process
   rtn = CPIA_GrabFrame(0);
   if ( !rtn ) {
      printf( "CPIA_GrabFrame failed\n" );
   }

     do {
          rtn = CPIA_GetCameraStatus( NULL, NULL, &streamState, NULL,
         NULL,        NULL, NULL, NULL );
          if ( !rtn ) {
             printf( "CPIA_GetCameraStatus failed\n" );
             break;
          }
     } while( streamState != STREAM_READY );

     // Transfer process
```

```
    rtn = CPIA_UploadStreamData( STREAM_BUFFER_LEN, gStreamBuffer,
  &actual_bytes, 1024 );
    if ( !rtn ) {
        printf( "CPIA_UploadStreamData failed\n" );
    }
   //printf( "UploadImage: Image uploaded %lu bytes\n", actual_bytes );
   return 1;
  }
```

Abbildung 23: SingleGrab( )

Die weitere Verarbeitung der Bilddaten wird unabhängig von der Kamera durchgeführt. Mit dem Aufruf von *verifyAndFormat()* wird das im Buffer abgelegte Frame dekomprimiert und in 24-Bit RGB-Werte transformiert. Dies ist nötig da die Bilder in einem PPM-Format gespeichert werden sollen. Das PPM-Format (= Portable Pixel Map) bietet sich an, da es sehr einfach aufgebaut ist. Im einfachsten Fall wird eine Textdatei angelegt und die RGB-Farbwerte hintereinander für jedes Pixel eingefügt. Zudem müssen zu Beginn der Datei eine Magic Number und die Bildabmessungen enthalten sein. Mit der Dateiendung .ppm ist es für die meisten Bildbetrachtungsprogramme dann keine Schwierigkeiten die Bilder anzuzeigen.

Abbildung 24: Erstes Bild ohne geänderter Pixelfolge und vertauschten RB-Werten

Mit der Methode *saveBufferToFile( )* werden die Bildinformationen im PPM-Format gespeichert. Nach dem Schreiben des korrekten Headers werden die RGB-Werte aus dem Buffer in die Datei eingefügt. Die ersten drei Werte im Buffer entsprechen dem BGR-Wert des letzten Pixels im Bild. Deshalb müssen die RGB Tripel von hinten nach vorne aus dem Buffer ausgelesen und dann mit vertauschten Rot- und

Blauwerten in die Datei geschrieben werden. Die obige Abbildung zeigt das Ergebnisbild ohne die Änderung der Pixelreihenfolge und der RB-Werte. Die grünen Farbwerte sind korrekt, aber der Blumentopf ist normalerweise rot und nicht blau.

```
void saveBufferToFile(void){
        FILE* imagefile;
        char* filename1="frame";
        char* filename="";
        char numberAsString[5];

        filename = new char[strlen(filename1)+12];
        strcpy(filename, filename1);
        if (continuousGrabbing){
            itoa(imageNumber,numberAsString,10);
//          printf(numberAsString);
            strcat(filename,numberAsString);
//          printf("%s\n",filename);
        }
        strcat(filename, ".ppm");
        imagefile = fopen( filename, "wb" );
        if ( imagefile == NULL ) {
            printf( "UploadImage: Unable to open %s file\n", filename );
            return;
        }
        //Header für PPM Format
        if (VIDEOSIZE==VIDEOSIZE_QCIF)
            fprintf(imagefile,"P3\n%d\n%d\n255\n",imageWidth/2,imageHeight/2);
        else
            fprintf(imagefile,"P3\n%d\n%d\n255\n",imageWidth,imageHeight);

        int pixel=0;
        int stopH=0, stopW=0;
        if (VIDEOSIZE==VIDEOSIZE_QCIF){
            stopH=imageHeight/2;
            stopW=imageWidth/2;
        }
        // Pixel richtig umsortiert in Datei schreiben. Für QCIF nur das letzte
        // Bufferviertel.
        for(int lines=imageHeight; lines>stopH; lines--) {
            for(int cols=imageWidth; cols>stopW; cols--){
                pixel=(imageWidth*3*lines)-(cols*3);
                fprintf( imagefile, "%u ",rgb24Buffer[pixel+2]);
                fprintf( imagefile, "%u ",rgb24Buffer[pixel+1]);
                fprintf( imagefile, "%u ",rgb24Buffer[pixel]);
            }
            fprintf(imagefile,"\n");
        }
        //printf("File written\n");
        fclose( imagefile );
}
```

Abbildung 25: SaveBufferToFile( )

```
void ContinousGrab(void){
    bool rtn;
    bool stop=false;

    continuousGrabbing=true;
    CPIA_GotoHiPower();
    CPIA_SetGrabMode(1);

    CPIA_SetFormat(VIDEOSIZE,subSampling, 0 );
    CPIA_SetCompression(compMode,decimation);
    long start=clock();
    do {
          SingleGrab(0,0,rtn);
          imageNumber ++;
          verifyAndFormat();
          saveBufferToFile();
          if (_kbhit()) stop = true;
          if (imageNumber>=1100)stop =true;
    } while(!stop);

    CPIA_GotoLoPower();
    printf( "%d images taken.\n", imageNumber-1000);
    long end= clock();
    printf( "Time: %d\n", (end-start)/1000);
    imageNumber=1000;
}
```

Abbildung 26: ContinousGrab( )

Für die kontinuierliche Bildaufnahme kann die Methode *ContinuousGrab( )* verwendet werden. Nach Übertragung der Aufnahmeparameter wird *SingleGrab( )* zusammen mit *verifyAndFormat( )* und *saveBufferToFile ( )* in einer Schleife aufgerufen. Die Schleife wird verlassen, sobald eine Taste gedrückt wird oder nachdem 100 Bilder aufgenommen worden sind. Dabei werden die Bilder mit fortlaufenden Nummern gespeichert.

Aufnahmezeiten für100 Bilder ohne Farbkonvertierung und Speichern der Dateien

Kompression	off	on	on	off
Decimation	off	on	off	on
QCIF 4:2:0	26s	4s	14s	20s
QCIF 4:2:2	36s	6s	20s	20s
CIF 4:2:0	106s	9s	37s	70s
CIF 4:2:2	139s	20s	47s	71s

Abbildung 27: Aufnahmezeiten für ContinuousGrab( )

Eine alternative Möglichkeit kontinuierlich Bilder aufzunehmen, besteht in der Nutzung von *StreamGrab( )*. Hierbei sendet die Kamera von sich aus automatisch ein Frame nach dem anderen. Es muss also nicht für jedes Einzelbild *SingleGrab( )* aufgerufen werden. *CPiA_SetGrabMode(1)* aktiviert den kontinuierlichen Aufnahmemodus. *CPIA_InitStreamCap( )* versetzt die Kamea in den Stream Capture Modus. In diesem Modus werden GrabFrame und UploadFrame Befehle (z.B. durch *SingleGrab()* ausgelöst) ignoriert. Erst wenn der Modus über *CPIA_FiniStreamCap()* wieder beendet wird, werden GrabFrame und UploadFrame Befehle wieder ausgeführt. Durch *CPIA_InitStreamCap( )* findet noch keine Aufnahme und Übertragung von Bildern statt. Erst durch *CPIA_StartStreamCap( )* beginnt die Kamera damit und überträgt solange Bilder, bis *CPIA_StopStreamCap( )* aufgerufen wird.
Durch den kontinuierlichen Upload der Bildinformationen müssen für die Weiterverarbeitung die einzelnen Bilder aus dem Datenstrom extrahiert werden. Der Beginn eines Frames wird durch die beiden Magic Numbers im 64 Byte langen Frameheader signalisiert. Das Ende eines Frames markieren vier aufeinanderfolgende 0xFF Werte. Falls die Übertragung weiterläuft, folgen weiter 0xFF Werte bis ein weiteres Frame beginnt.[1]

```
void StreamGrab(void){
        bool rtn;
        rtn=CPIA_SetGrabMode(1);

        rtn=CPIA_InitStreamCap(0,0);

        rtn=CPIA_StartStreamCap();

        do {
        } while( !_kbhit() );

        rtn=CPIA_StopStreamCap();

        rtn=CPIA_FiniStreamCap();
}
```

Abbildung 28: StreamGrab( )

5.2.5 pp_test.c

Pp_test.c stellt Methoden bereit, um die Funktionalität der Kamera und des CPiA-Controllers zu testen. Über den Menüpunkt Systemtest im Benutzermenü wird die Methode *DoSystemTest( )* gestartet. DoSystemTest( ) ruft die Versionen der Firmware und des Videocontrollers ab, überprüft den Status der Kamera und überträgt Daten im

[1] Vision CPiA Data Sheet.pdf Kap. 3.9.3 End of Frame / 3.11.1 Description of ECP / Nibble Upload Modes.

Nibble- und im ECP-Modus von der Kamera zu Computer. Weiterhin werden der Videoprozessor und das EEPROM der Kamera getestet.

5.3 SmallVersionECP

SmallVersionECP ist eine Folgeversion von PP_Test. Es bietet annähernd die gleichen Funktionen, ist aber im Umfang des Quellcodes deutlich reduziert. Die größten Änderungen betreffen cpia_cmd.c, cpia_link.c und decode.c. Hier wurden zu Gunsten der Übersichtlichkeit viele Methoden, die nicht gebraucht werden, aus dem Quellcode entfernt. Gravierende Änderungen an der Funktionsweise gibt es jedoch nicht. Die Bedienung des Programms erfolgt wie bei PP_Test über ein Textmenü.

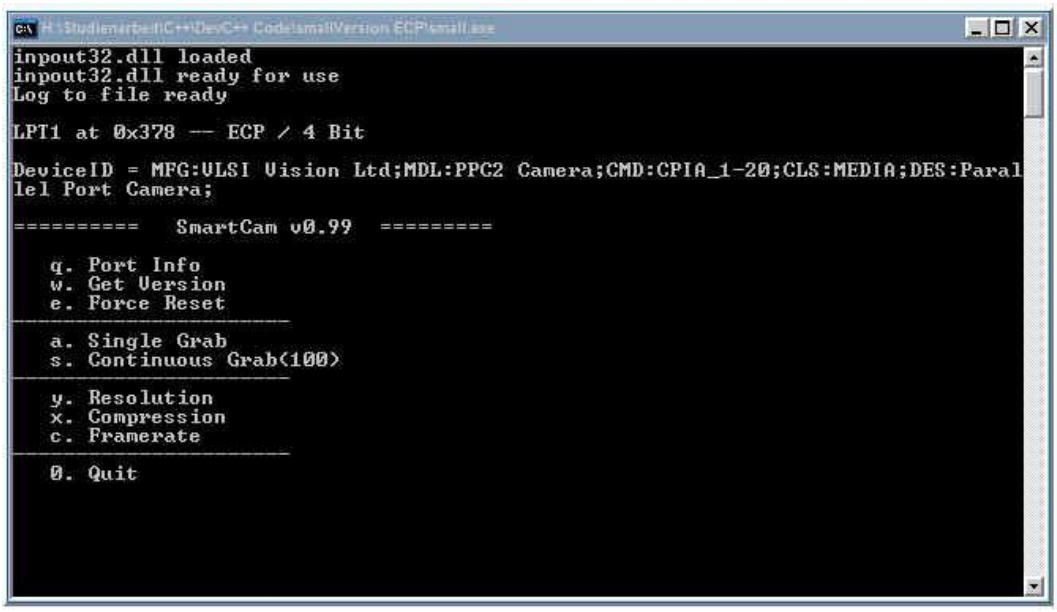

Abbildung 29: SmallVersionECP Benutzermenü

5.4 CamMonitor

CamMonitor ist ein Programm zur Bedienung der Creative VideoBlaster II Webcam mit einer grafischen Benutzeroberfläche. Das Programm bietet die Möglichkeit die Bildübertragung der Kamera per Mausklick zu starten und als Livestream im Fenster zu betrachten. Während des Betriebs können zudem das Bildformat und die Kompression geändert werden.
Die Oberfläche wurde mit einer Testversion der GUI Bibliothek Qt 4.0 von Trolltech[1] realisiert. Basis für die Ansteuerung der Kamera ist SmallVersionECP, auf das die GUI aufbaut. Die für die Darstellung der GUI relevanten Daten sind in den Dateien imageviewer.cpp und thread.cpp enthalten. Das Layout der Programmoberfläche wird von

[1] http://www.trolltech.com

Qt in der Datei imageviewer.ui verwaltet. Da sich die Qt 4.0 Version direkt in das Visual Studio integriert, lassen sich die Oberflächen dort per Drag & Drop erstellen. Änderungen an der erstellten Oberfläche werden beim normalen Kompilieren ebenfalls mit eingebunden, so dass die Oberflächen nicht mehr außerhalb vom Visual Studio kompiliert werden müssen. Für die Ausführung einer mit Visual Studio und Qt erstellen Releaseversion, müssen neben der inpout32.ddl auch qtgui4.dll, qtcore4.dll, msvcp71.dll und msvcr71.dll im gleichen Verzeichnis vorhanden sein. Ist dies nicht der Fall oder sind keine Pfade zu den dll-Dateien gesetzt, bricht das Programm mit einer entsprechend Fehlermeldung ab.

5.4.1 imageviewer.cpp

Die Datei imageviewer.cpp enthält die gleichnamige Klasse imageviewer und ist das Herzstück der GUI. Hier werden die verschiedenen Oberflächenelemente, z.B. Pushbuttons und Radiobuttons mit ihren Eigenschaften beschrieben und gruppiert. Zudem werden die dort sogenannte Slots und Actions angelegt. Signale brauchen nicht explizit angelegt zu werden. Knöpfe versenden z.B. automatisch das Signal clicked, wenn sie angeklickt werden. Slots sind die angelegten Klassenmethoden die zur Ausführung bereit stehen. Actions können z.B Einträge in der Menüstruktur sein. Über ein *connect( )* können Actions, Oberflächenelemente und Signale mit Slots verbunden werden.

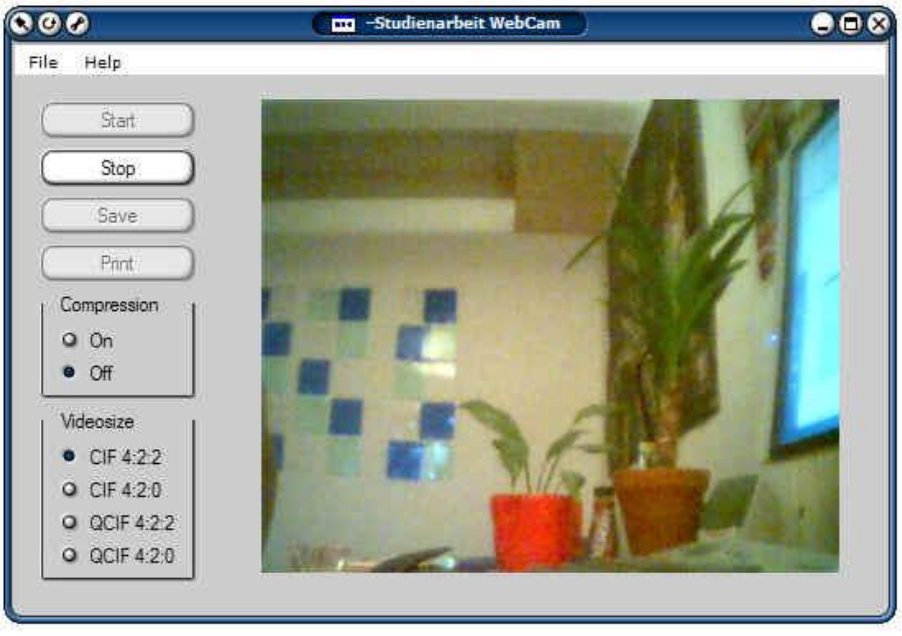

Abbildung 30: CamMonitor mit grafischer Benutzeroberfläche

5.4.2 thread.cpp

Nach dem Start des Programms wird von der main-Methode die GUI gestartet. Mit einem Klick auf den Startknopf oder Start im Filemenü wird der Slot *start( )* aufgerufen. Hier wird zunächst die Kamera an einem der drei LPT-Ports gesucht und mit *startCamControl( )* der Datenaustausch mit der Kamera vorbereitet. Nach erfolgreicher Initialisierung öffnet sich ein Fenster mit den Versionsdaten der Kamerafirmware und des Videoprozessors. Im nächsten Schritt wird dann der Bildaufnahmethread gestartet und die Zustände der Bedienelemente aktualisiert.
Ist der Thread gestartet, nimmt er kontinuierlich Bilder auf. Die Bildinformationen werden dekomprimiert und in RGB-Werte transformiert. Aus den RGB-Werten wird schließlich ein QImage erstellt. Sobald dies geschehen ist, wird ein Signal ausgelöst, das den Slot *display(const QImage &image)* startet und die Darstellung des QImages in der Oberfläche aktualisiert.

```
Thread::Thread(QObject *parent) : QThread(parent){
        restart = false;
        abort = false;
 }

void Thread::grabImage(){
        if (!isRunning()) {
                start(NormalPriority);
        } else  exit();
}

void Thread::stop(){
         abort=true;
}
void Thread::run(){
        while (!abort){
                int stopH=0;
                int stopW=0;
                int lines=288;
                int cols=352;
                unsigned int rgb;
                int bufferPos;
                int stop=0;
                bool rtn;
                /* get Image */
                singleGrab();

                if (getVideoSize()){
                        lines=144;
                        cols=176;
                        stopH=144;
                        stopW=176;
                }
                QImage image(cols,lines,QImage::Format_RGB32);
                /* get Image Data */
```

```
                uchar* buffer= getBuffer();
                /* sort Data and set Pixel in output Image */
                for(lines=288; lines>stopH; lines--) {
                        for(cols=352; cols>stopW; cols--){
                                bufferPos=(352*3*lines)-(cols*3);
                                rgb=qRgb(buffer[bufferPos+2],buffer[bufferPos
                                                                +1],buffer[bufferPos];
                                image.setPixel(352-cols,288-lines,rgb);
                        }
                }
                emit imageGrabbed(image);
  }
 }
```

Abbildung 31: Thread zur Bildaufnahme

5.5 XsmallVersion

Eine weitere Programmversion zur Ansteuerung der VideoBlaster II Webcam trägt den Namen XsmallVersion. Bei dieser Version wird die Kamera sehr hardwarenah angesprochen. Die Steuerbefehle werden direkt an den Parallelport gesendet. Das Auslesen des Ports erfolgt ebenso direkt. Bei den anderen Version lief die Steuerung der Kamera über die Methoden in pport.c , cpia_link.c, und cpia_cmd.c. Zusätzlich waren noch die Befehlsdefinitionen in cmd.h nötig. In der XsmallVersion sind alle nötigen Befehle und Methoden zur Steuerung sowie Datenübertragung in der Datei camera.c enthalten.

Wiederholung des Aufbaus der Steuerbefehle:
Alle CPiA-Steuerbefehle bestehen aus einer Folge von acht Byte. Das erste Byte (bmRequestType) legt fest, ob eine Antwort von Gerät erwartet wird. Bei einem Wert 0xC0 ist dies der Fall (device to host transfer). Ist der Wert 0x40 wird von der Kamera keine Antwort erwartet (host to device transfer). Eine Ausnahme bilden hier die Befehle, die eine Übertragung von Bilddaten bewirken. Das zweite Byte (bRequest) identifiziert den Befehl eindeutig. Die nächsten vier Byte dienen der Übergabe von notwendigen Parametern. Das vorletzte Byte legt fest wie viele Bytes als Antwort auf den Befehl zu senden sind. Dieser Wert wird nur gesetzt, falls ein informationsanfordernder Befehl (device to host Transfer) vorliegt. Bedeutung der vom CPiA-Chip zurückgesendeten Bytes ist fest definiert.

bmRequestType	bRequest	wValue		wIndex		wLength	
11000000	000 00001	0x00	0x00	0x00	0x00	0x04	0x00

Abbildung 32: Der Befehl GetCPiAVersion

Der oben aufgeführte Befehl GetCPiAVersion wird also durch die folgenden acht Byte repräsentiert: {0xC0, 0x01, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00}.
Für eine Ansteuerung der Kamera ohne großen Overhead müssen also in der Theorie lediglich Folgen von acht Bytes an den CPiA-Chip gesendet werden können. Diese Schreiboperation sollte gemäß Datenblatt[1] im ECP-Modus durchgeführt werden. Für den Empfang der Kameradaten muss der Parallelport Byteweise ausgelesen werden können. Für die Leseoperation kann der Datenübertragungsmodus vorher auf ECP- oder Nibble-Modus festgelegt werden. Nach diesem Prinzip funktionieren auch die anderen Programmversionen, allerdings sehr viel verschachtelter als in camera.c .

5.5.1 SendPacket( )

Um einen Befehl, z.B. GoToHiPower, an den CPiA-Chip zu senden, wird SendPacket mit dem entsprechenden Befehl als Parameter aufgerufen. In diesem Fall SendPacket(g_ucPktGotoHiPower). *SendPacket ( )* führt zuerst ein *TransferMode(REQUEST_ECP)* aus. Damit wird der Übertragungsmodus auf ECP gesetzt. Dann wird die Übertragung des Befehls mit *WriteBuffer( )* ausgeführt.
WriteBuffer( ) überträgt das übergebene Befehlspacket Byte für Byte an die Kamera. Abschließend wird der Transfer mit *TransferEnd( )* beendet.

```
void SendPacket(unsigned char* packet)
{
        TransferMode(REQUEST_ECP);
        WriteBuffer( packet, 8);
        TransferEnd();
}
```

Abbildung 33: SendPacket( )

```
int WriteBuffer( unsigned char *Data, unsigned long Bytes )
{
  unsigned long i,c;
  unsigned char *ptr;

  ECP_AUTOFEED_OFF;

  /* we are now in forward idle phase */
  while( ECP_BUSY == true );

  ptr = Data;
  for( i = 0; i < Bytes; i++ )
  {
        set_ECP_DATA(Data[i]);
```

[1] cpia_datasheet.pdf Kap. 2.2.5.8 Command / Status Transfers

```
//usleep(2000);
                    ECP_STROBE_OFF;
//usleep(2000);
            while( ECP_BUSY == false ); // event 36

                    ECP_STROBE_ON;
            while( ECP_BUSY == true ); // event 36
    }

    return( true );
}
```

Abbildung 34: WriteBuffer( )

5.5.2 GetPacket( )

GetPacket( ) liest Daten von der Kamera und schreibt sie in einen per Verweis übergebenen Buffer. Um für die weitere Verwendung des Programms alle Möglichkeiten offen zu halten, sind die Leseoperationen sowohl im ECP-Modus als auch im Nibble-Modus realisiert. *GetPacket( )* wird direkt im Anschluss an *SendPackage()* verwendet, wenn der CPiA-Befehl (Status-)Daten von der Kamera anfordert. Analog zu *SendPacket( )* wird nach dem Aufruf erst der Transfermodus festgelegt. Dementsprechend wird entweder *NibbleRead Buffer( )* oder *ECPReadBuffer( )* aufgerufen. Die beiden Methoden lesen jeweils so viele Byte, wie es der übergebene Parameter Bytes vorgibt. Dabei muss die Methode *NibbleReadBuffer()* beachten, dass jedes Byte in zwei Hälften übertragen wird.
Nach dem Einlesen der Daten wird die Ausführung von *GetPacket( )* durch *TransferEnd( )* abgeschlossen.

```
int ECPReadBuffer(int uploadMode, unsigned char *Data, unsigned long Bytes){
        int i,c;

        if(!uploadMode)  {
                ECP_DATA_IN;
                ECP_INIT_OFF; // event 39
        }
        for( i = 0; i < Bytes; i++ )  {
                set_ECP_CONTROL(0x22);
                while( ECP_ACK == true ); // event 43

        Data[i] = get_ECP_DATA;
        set_ECP_CONTROL(0x20);
        while( ECP_ACK == false ); //event 45

        }
        if(!uploadMode)
                ECP_INIT_ON; // event 39
        return(1);
}
```

Abbildung 35: ECPReadBuffer( )

5.5.3 UploadStreamData( )

UploadStreamData( ) wird zum Empfangen der Bilddaten verwendet. Zur Aufnahme eines Bildes wird zuvor ein *SendPacket(g_ucPktGrabFrame)* an die Kamera gesendet. Mit dem Befehl *Send-Packet(g_ucPktGetStatus)* wird festgestellt, wann die Kamera zur Übertragung der Bildinformationen bereit ist. Mit *UploadStreamData( )* werden die Bildaten dann byteweise mit *NibbleReadBuffer( )* oder *ECPReadBuffer( )* gelesen. Mit dem Finden der EOF(End of Frame), einer Folge von vier 0xFF Werten, wird der Lesevorgang beendet. Das aufgenommene Bild kann dann wie gewohnt dekomprimiert und gespeichert werden.

```
int UploadStreamData( int EndOnFF, unsigned long DataBytes, unsigned char
   *DataBuffer, unsigned long *ActualBytes, unsigned long StreamChunkSize )
{
   unsigned long curr;

   TransferMode(REQUEST_LINK);
   ECP_DATA_IN;

   for( curr = 0; curr < DataBytes; curr += StreamChunkSize )
   {
         #ifdef NIBBLE_MODE
               NibbleReadBuffer( true, &DataBuffer[curr], StreamChunkSize );
         #else
               ECPReadBuffer( true, &DataBuffer[curr], StreamChunkSize );
         #endif

         *ActualBytes = curr + StreamChunkSize;
          if ( (DataBuffer[curr+StreamChunkSize-4] == 0xFF)
            && (DataBuffer[curr+StreamChunkSize-3] == 0xFF)
            && (DataBuffer[curr+StreamChunkSize-2] == 0xFF)
            && (DataBuffer[curr+StreamChunkSize-1] == 0xFF) ) break;
   }
   ECP_INIT_ON;
   if(!TransferEnd())return(0);
   return( true );
}
```

Abbildung 36: UploadStreamData( )

6 Fazit und Ausblick

Die Arbeit an der Studienarbeit war sehr vielseitig und lehrreich. Ich konnte mich in Themenbereiche einarbeiten, mit denen ich mich noch nicht befasst hatte, z.B. die Oberflächengestalltung mit Qt. Auch das Überwinden der kleineren und größeren Probleme bis zur Aufnahme des ersten Bildes mit der Webcam war ein kontinuierlicher Prozess. Generell können die zahlreichen Foren im Internet zur Lösung von Problemen sehr hilfreich sein. Besonders zur Thematik der Ansteuerung des Parallelports unter WindowsXP hat die Suche in den Foren gute Hinweise geliefert. Besonders hervorheben möchte ich an dieser Stelle die Foren supportnet.de, mikrocontroller.net und tutorials.de. Natürlich aber auch die gute Betreung von Dr. Merten Jost.

Nachdem die Ansteuerung der Creative VideoBlaster II Webcam mit den verschieden Programmversionen funktioniert, ist besonders die vierte Version für eine direkte Ansteuerung der Kamera mit einem Controller zur Realisierung einer Netcam interessant. CamMonitor und XsmallVersion werde ich auch nach dem Ende der Studienarbeit weiterentwickeln und weitere Funktionen hinzufügen.

Literatur

1 Dirk Louis. C/C++ Kompendium – Das komplette Programmierwissen für Studium und Job. Markt und Technik Verlag, München, 2003

2 Klaus Dembowski. PC-Werkstatt. Markt und Technik Verlag, München, 2000

3 VISION CPiA Data Sheet (VV0670P001), Colour Processor Interface ASIC, 02/07/98

4 Software Developer's Guide for CPiA Cameras, Release 1.0, 1998

5 IEEE P1284. IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers, Draft D1.2, August 1, 2000

6 Creative VideoBlaster WebCam II Benutzerhandbuch

7 http://webcam.sourceforge.net/

8 http://sourceforge.net/projects/cpiawindows

9 http://www.logix4u.net/inpout32.htm

10 http://java.sun.com/products/javacomm/

11 http://www.geocities.com/Juanga69/parport

12 http://www.naughter.com

13 http://neil.fraser.name/software/ltp/

14 http://www.trolltech.com

15 http://www.supportnet.de

16 http://www.mikrocontroller.net

17 http://www.tutorials.de.